Lenguajes Formales y Computabilidad
Resumen Teorico
Parcial 4
Guias 7 a 9

Agustín M. Domínguez

1C 2025

Índice

Este apunte tiene las definiciones importantes de la materia, sin varias de las explicaciones o demostraciones que están en las guías, a modo de *resumen*.

Esta materia tiene la particularidad que (aparte de las clases mismas) las guías son el recurso más completo para el estudiante, que funcionan como **Resumen de la clase**, **Apunte Teórico**, y **Práctico de la materia** en un solo documento. Otra particularidad es que las guías son documentos vivos, en el sentido que los profes están constantemente expandiendo y ajustando el contenido, por lo que este documento *puede* estar desactualizado si se consulta en el futuro. Siempre conviene buscar la última versión de las guías y estudiar desde ahí. Sin embargo, este documento puede servir para tener las definiciones importantes de la materia a mano.

Guía 7: El Paradigma de Neumann (Parte 2)

1.1 Funciones Σ-computables

Def: **Notación de estados**

Dados $x_1, \ldots, x_n \in \omega$ y $\alpha_1, \ldots, \alpha_m \in \Sigma^*$ con $n, m \in \omega$ usamos:

$$\|x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m\|$$

para denotar el estado:

$$((x_1, \ldots, x_n, 0, 0, \ldots), (\alpha_1, \ldots, \alpha_m, \varepsilon, \varepsilon, \ldots))$$

Esto es simplemente una notación para ahorrar tener que escribir todo el tiempo las dos infinituplas de un estado.

Def: **Función 'horquilla'** (Ψ)

Dado $\mathcal{P} \in \text{Pro}^{\Sigma}$, definiremos para cada $n, m \geq 0$ la función:
$\Psi_{\mathcal{P}}^{n,m,\#}$ y la función $\Psi_{\mathcal{P}}^{n,m,*}$ de la siguiente manera:

$$D_{\Psi_{\mathcal{P}}^{n,m,O}} = \{(\overrightarrow{x}, \overrightarrow{\alpha}) \in \omega^n \times \Sigma^{*m} \ : \ \mathcal{P} \text{ termina partiendo de } \|x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m\|\}$$

con $O \in \{\#, *\}$

$\Psi_{\mathcal{P}}^{n,m,\#}(\overrightarrow{x}, \overrightarrow{\alpha})$ = valor de N1 en el estado obtenido cuando $\mathcal{P}$ termina*

$\Psi_{\mathcal{P}}^{n,m,*}(\overrightarrow{x}, \overrightarrow{\alpha})$ = valor de P1 en el estado obtenido cuando $\mathcal{P}$ termina*

(*) Termina partiendo de $\|x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m\|$

Def: **Funciones Σ-computables**

Una función Σ-mixta $f : S \subseteq \omega^n \times {\Sigma^*}^m \to \omega$ será llamada Σ-computable si hay un programa $\mathcal{P}$ de $\mathcal{S}^\Sigma$ tal que $f = \Psi_{\mathcal{P}}^{n,m,\#}$
Análogamente, una función Σ-mixta $f : S \subseteq \omega^n \times {\Sigma^*}^m \to \Sigma^*$ será llamada Σ-computable si hay un programa $\mathcal{P}$ de $\mathcal{S}^\Sigma$ tal que $f = \Psi_{\mathcal{P}}^{n,m,*}$
En cualquier caso caso diremos que la función f es *computada* por $\mathcal{P}$

Proposición: **Leibniz vence a Neumann**

Si f es Σ-computable, entonces f es Σ-efectivamente computable.

1.2 Macros

Esta es la herramienta más poderosa que tiene el paradigma imperativo de Neumann ya que permite crear programas de más alto nivel a partir de otros.

La idea básica del macro es escribir una palabra 'atajo' que reemplaza muchas instrucciones que realizan un procedimiento sin efectos secundarios, para poder agrupar comportamiento más complejo que las instrucciones básicas del lenguaje no permiten fácilmente.

Un macro se conforma de dos partes: Primero la declaración que consta de una palabra de la forma:

$$\left[VK_1 \leftarrow f(VK_2, VK_3, \ldots, WK_4, WK_5, \ldots)\right]$$

o de la forma

$$\left[WK_1 \leftarrow f(VK_2, VK_3, \ldots, WK_4, WK_5, \ldots)\right]$$

y el otro tipo de macro tiene la forma:

$$\left[\text{IF } P(VK_1, VK_2, \ldots, WK_3, WK_4, \ldots) \text{ GOTO } AK_5\right]$$

Que representa la "llamada" al macro.

Y luego tenemos el "molde" que es parecido a un programa que define la el código del macro. Este es un programa normal de Pro^Σ con dos cambios.

Igual que en la declaración, las letras de las variables cambian en el macro.

- Variables numéricas se escriben con $V\overline{n}$ en lugar de $N\overline{n}$
- Variables alfabéticas se escriben con $W\overline{n}$ en lugar de $P\overline{n}$
- Labels se escriben con $A\overline{n}$ en lugar de $L\overline{n}$

El segundo cambio es que se permite referirse a una label no presente en el programa mismo, si esta es una label "oficial" del macro.

Estas variables se van a reemplazar en el programa madre de tal manera que **no se pisen variables auxiliares para evitar efectos secundarios**

> *Def:* **Variables oficiales**
> Son las variables numéricas, alfabéticas, o labels que aparecen como argumentos en la declaración del macro

> *Def:* **Variables auxiliares**
> Son las variables numéricas, alfabéticas, o labels que no aparecen como argumentos en la declaración del macro pero son usadas en el programa que lo define.

Por ejemplo tenemos el siguiente macro:

$$\big[V1 \leftarrow \lambda xy[x + y](V2, V3)\big]$$

Definido como:

```
V4←V2
A1 IF V4≠0 GOTO A2
GOTO A3
A2 V4←V4−̇1
V1←V1+1
GOTO A1
A3 SKIP
```

Tenemos varias cosas que notar:

- El molde del macro claramente escribe en V1 el resultado de la función esperada.
- V1, V2, y V3 son variables oficiales
- A1,A2,A3, y V4 son variables auxiliares.
- Este programa no modifica el valor de ninguna variable oficial, exceptuando la variable V1 que está a la izquierda de la flecha. Esta es una condición **obligatoria** para considerarse un macro válido.

Un ejemplo de como se va a instanciar el macro puede ser el siguiente:

$\left[\texttt{N1} \leftarrow \lambda xy[x+y](N1, N2)\right]$
$\left[\texttt{N1} \leftarrow \lambda xy[x+y](N1, N3)\right]$
$\left[\texttt{N1} \leftarrow \lambda xy[x+y](N1, N4)\right]$
`IF N1≠0 GOTO L1`
`N1←N1+1`
`L1 N1←0`

Este es un programa arbitrario que básicamente computa el predicado

$\lambda wxyz[w = 0 \wedge x = 0 \wedge y = 0 \wedge z = 0]$

Def: **Macro asociado a una función Σ-computable**
Dada una función $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to \omega$ usaremos

$$\left[V\overline{n+1} \leftarrow f(V1, \ldots, V\overline{n}, W1, \ldots W\overleftarrow{m})\right]$$

para denotar un macro M el cual cumpla las propiedades de un programa que computa f en el sentido que si se reemplaza correctamente en su llamada por variables de $N\overline{k_1}, \ldots P\overline{k_{n+m}}$ y sus variables auxiliares y labels por variables no usados en el programa madre y distintos entre sí uno a uno, entonces se crea un programa que se detiene cuando la entrada pertenece al dominio de f y devuelve la evaluación de la función con esos inputs, y cuando dicho input no pertenezca al dominio de la función no se detenga.
La explicación más detallada y técnica está en la guía, pero intuitivamente hablando el macro dice que se guardará en $V\overline{n+1}$ el resultado de evaluar la función f con los argumentos llamados si está en el dominio, o no se va a detener si no lo está.

Proposición: **Existencia de Macros**
Sea Σ un alfabeto finito.
(a) Sea $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to \omega$ una función Σ-computable. Entonces en $\mathcal{S}^\Sigma$ existe un macro:

$$\left[V\overline{n+1} \leftarrow f(V1, \ldots, V\overline{n}, W1, \ldots W\overleftarrow{m})\right]$$

(b) Sea $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to \Sigma^*$ una función Σ-computable. Entonces en $\mathcal{S}^\Sigma$ existe un macro:

$$\left[W\overline{n+1} \leftarrow f(V1, \ldots, V\overline{n}, W1, \ldots W\overleftarrow{m})\right]$$

(c) Sea $P : D_P \subseteq \omega^n \times \Sigma^{*m} \to \omega$ un **predicado** Σ-computable. Entonces en $\mathcal{S}^\Sigma$ existe un macro:

$$\left[\text{IF } P(V1, \ldots, V\overline{n}, W1, \ldots W\overleftarrow{m}) \text{ GOTO A1}\right]$$

1.3 Conjuntos Σ-enumerables y Σ-computables

Def: **Conjunto Σ-enumerable**

Un conjunto $S \subseteq \omega^n \times \Sigma^{*m}$ será llamado Σ-enumerable cuando sea vacío o haya una función $F : \omega \to \omega^n \times \Sigma^{*m}$ tal que $I_F = S$ y $F_{(i)}$ sea Σ-computable, para cada $i \in \{1, \ldots, n+m\}$

Proporción: **Caracterizción de Σ-enumerabilidad**

Sea $S \subseteq \omega^n \times \Sigma^{*m}$ un conjunto no vacíol. Entonces son equivalentes:

- S es Σ-enumerable
- Existe un programa $\mathcal{P} \in Pro^{\Sigma}$ tal que:

  Para cada $x \in \omega$, tenemos que $\mathcal{P}$ se detiene partiendo del estado $\|x\|$ y llega a un estado de la forma $((x_1, \ldots, x_n, y_1, \ldots), (\alpha_1, \ldots, \alpha_m, \beta_1, \ldots)) \in S$ —

  Para cada $(x_1, \ldots, x_n, \alpha_1, \alpha_m) \in S$ existe un $x \in \omega$ tal que $\mathcal{P}$ se detiene partiendo desde el estado $\|x\|$ y llega a un estado de la forma $((x_1, \ldots, x_n, y_1, \ldots), (\alpha_1, \ldots, \alpha_m, \beta_1, \ldots))$

Def: **Conjuntos Σ-computables**

Un conjunto $S \subseteq \omega^n \times \Sigma^{*m}$ será llamado Σ-computable cuando la función $\chi_S^{\omega^n \times \Sigma^{*m}}$ sea Σ-computable

Si $\mathcal{P}$ es un progama el cual computa $\chi_S^{\omega^n \times \Sigma^{*m}}$, diremos que $\mathcal{P}$ decide la pertenencia a S con respecto al conjunto $\omega^n \times \Sigma^{*m}$

Def: **Macros definidos a conjuntos Σ-computables**

Definiremos una nueva **notación**

Si existe el macro:

`[IF` $\chi_S^{\omega^n \times \Sigma^{*m}}$`(V1,. . . ,` $V_{\overline{n}}$`, W1,. . . ,` $W_{\overline{m}}$`) GOTO A1]`

lo podremos escribir como:

`[IF (V1,. . . ,` $V_{\overline{n}}$`, W1,. . . ,` $W_{\overline{m}}$`)`$\in S$ `GOTO A1]`

Notar que la condición para usar el segundo macro es que exista el primero, es decir en cierta forma que se pueda computar la pertenencia, o dicho de otra forma que S sea compuble

Proposición: **Primer manancial de macros**

En esta materia asi le decimos a la idea de que, gracias a la proposición anterior, podemos tomar cualquier función que ya demostramos es Σ-computable y convertirla en un macro para crear otros programas. Es un "manantial" porque es una fuente de la cual podemos sacar macros.
Sea Σ un alfabeto finito. Si:

$$f \ : \ D_f \subseteq \omega^n \times {\Sigma^*}^m \to \omega$$

$$g \ : \ D_g \subseteq \omega^n \times {\Sigma^*}^m \to \Sigma^*$$

$$P \ : \ D_P \subseteq \omega^n \times {\Sigma^*}^m \to \{0,1\}$$

son Σ-**computables**, entonces en $\mathcal{S}^{\Sigma}$ existen los macros:
`[`$V_{\overline{n+1}} \leftarrow f$`(V1,. . . ,`$V_{\overline{n}}$`,W1,. . . ,`$W_{\overline{m}}$`)]`
`[`$V_{\overline{m+1}} \leftarrow g$`(V1,. . . ,`$V_{\overline{n}}$`,W1,. . . ,`$W_{\overline{m}}$`)]`
`[IF P(V1,. . . ,` $V_{\overline{n}}$`, W1,. . . ,` $W_{\overline{m}}$`) GOTO A1]`

Guía 8: Batalla entre Paradigmas

En esta guia comparamos los tres paradigmas de computabilidad efectiva que vimos (Turing, Gödel y Neumann), y demostraremos que los 3 son equivalentes, es decir que la cantidad de funciones que incluyen son iguales.

2.1 Neumann vence a Gödel

Theorem: **Neumann vence a Gödel**
Si h es Σ-recursiva, entonces h es Σ-computable

Proposición: **Segundo Manantial de Macros**
Sea Σ un alfabeto finito. Si:

$$f \ : \ D_f \subseteq \omega^n \times {\Sigma^*}^m \to \omega$$
$$g \ : \ D_g \subseteq \omega^n \times {\Sigma^*}^m \to \Sigma^*$$
$$P \ : \ D_P \subseteq \omega^n \times {\Sigma^*}^m \to \{0,1\}$$

son Σ-**recursivas**, entonces en $\mathcal{S}^\Sigma$ existen los macros:
`[`$V_{\overline{n+1}} \leftarrow f$`(V1,...,`$V_{\overline{n}}$`,W1,...,`$W_{\overline{m}}$`)]`
`[`$V_{\overline{m+1}} \leftarrow g$`(V1,...,`$V_{\overline{n}}$`,W1,...,`$W_{\overline{m}}$`)]`
`[IF P(V1,...,` $V_{\overline{n}}$`, W1,...,` $W_{\overline{m}}$`) GOTO A1]`

Lemma: **Union de conjuntos enumerables**
Sea $S_1, S_2 \subseteq \omega^n \times {\Sigma^*}^m$ conjuntos Σ-enumerables, entonces $S_1 \cup S_2$ y $S_1 \cap S_2$ son Σ-enumerable

Lemma: **Computable implica enumerable**
Si $S \subseteq \omega^n \times {\Sigma^*}^m$ es Σ-computable, entonces S es Σ-enumerable

2.2 Gödel vence a Neumann

Def: **Funciones $i^{n,m}, E_{\#}^{n,m}$, y $E_{*}^{n,m}$**
Definiremos conjuntamente las siguientes funciones:

$$i^{n,m} \ : \ \omega \times \omega^n \times \Sigma^{*m} \times Pro^{\Sigma} \to \omega$$
$$E_{\#}^{n,m} \ : \ \omega \times \omega^n \times \Sigma^{*m} \times Pro^{\Sigma} \to \omega^{[N]}$$
$$E_{*}^{n,m} \ : \ \omega \times \omega^n \times \Sigma^{*m} \times Pro^{\Sigma} \to \Sigma^{*[N]}$$

Sea $A = (0, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$
y $B = (t+1, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$
y $C = (t, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$
Entonces definimos las funciones tal que:
Caso Base:

$$(i^{n,m}(A), E_{\#}^{n,m}(A), E_{*}^{n,m}(A)) = (1, \quad (x_1, \ldots, x_n, 0, 0, \ldots), (\alpha_1, \ldots, \alpha_m, 0, 0, \ldots))$$

Caso Recursivo:

$$(i^{n,m}(B), E_{\#}^{n,m}(B), E_{*}^{n,m}(B)) = S_P(i^{n,m}(C), E_{\#}^{n,m}(C), E_{*}^{n,m}(C))$$

Definamos también:

$$E_{\#j}^{n,m} \ : \ \omega \times \omega^n \times \Sigma^{*m} \times Pro^{\Sigma} \to \omega$$
$$E_{*j}^{n,m} \ : \ \omega \times \omega^n \times \Sigma^{*m} \times Pro^{\Sigma} \to \Sigma$$

De la forma:
$E_{\#j}^{n,m}(t, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$ = j-esima coordenada de $E_{\#}^{n,m}(t, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$
$E_{*j}^{n,m}(t, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$ = j-esima coordenada de $E_{*}^{n,m}(t, \overrightarrow{x}, \overrightarrow{y}, \mathcal{P})$

Intuitivamente, dichas funciones guardan partes de la computación de $\mathcal{P}$ en el tiempo. i guarda la instrucción que se va a ejecutar luego de ejecutar cierta cantidad de pasos, y E_* y $E_\#$ guardan el estado alfabético y numérico respectivamente luego de ejecutar cierta cantidad de pasos. Es decir que si quisiera saber cual será el estado numérico luego de ejecutar 30 instrucciones a partir de un input neutro $(0, 0, \ldots, \varepsilon, \varepsilon, \ldots)$, ese resultado será:

$E_{\#}(30, 0, \ldots, \varepsilon, \ldots, \mathcal{P})$

Proposición:
Sean $n, m \geq 0$. Las funciones $i^{n,m}, E_{\#}^{n,m}$, y $E_{*}^{n,m}$ son $(\Sigma \cup \Sigma_P)$-PR

Def: $\text{Halt}^{n,m}$

$\text{Halt}^{n,m} = \lambda t \overline{x} \overline{\alpha} \mathcal{P}\big[i^{n,m}(t, \overline{x}, \overline{\alpha}, \mathcal{P}) = n(\mathcal{P}) + 1\big]$

Intuitivamente $\text{Halt}^{n,m}(t, \overline{x}, \overline{\alpha}, \mathcal{P})$ es 1 si y solo si el programa $\mathcal{P}$ se detiene luego de t pasos partiendo desde el estado $\|x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m\|$

Lemma:

- Pro^{Σ} es un conjunto $(\Sigma \cup \Sigma_P)$-PR
- $\lambda \mathcal{P}[n(\mathcal{P})]$ y $\lambda i \mathcal{P}[I_i^{\mathcal{P}}]$ son funciones $(\Sigma \cup \Sigma_P)$-PR
- $\text{Halt}^{n,m}$ es $(\Sigma \cup \Sigma_P)$-PR

Def: $T^{n,m}$

$$T^{n,m} = M(\text{Halt}^{n,m})$$

Proposición:

$T^{n,m}$ es $(\Sigma \cup \Sigma_P)$-recursiva.

Teorema: **Gödel vence Neumann**

Si $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to O$ es Σ-computable, entonces f es Σ-recursiva

Proposición:

$T^{n,m}$ **NO** es $(\Sigma \cup \Sigma_P)$-PR.

2.3 Gödel vence a Turing

Teorema: **Gödel vence Turing**

Si $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to O$ es Σ-turing computable, entonces f es Σ-recursiva

2.4 Turing vence a Neumann

Teorema: **Turing vence a Neumann**

Si $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to O$ es Σ-computable, entonces f es Σ-turing computable

2.5 Tesis de Church

Teorema:
Sea Σ un alfabeto finito. Dada una función f, las siguientes afirmaciones son equivalentes:

- f es Σ-Turing computable
- f es Σ-recursiva
- f es Σ-computable

Teorema:
Sea Σ un alfabeto finito. Dado un conjunto $S \subseteq \omega^n \times \Sigma^{*m}$, las siguientes afirmaciones son equivalentes:

- S es Σ-Turing computable
- S es Σ-recursiva
- S es Σ-computable

Sacado de la guía:

Existe modelo matematico de computabilidad efectiva es el llamado ***lambda calculus****, introducido por Church, el cual tambien resulta equivalente a los estudiados por nosotros. El hecho de que tan distintos paradigmas computacionales hayan resultado equivalentes hace pensar que en realidad los mismos han tenido exito en capturar la totalidad de las funciones Σ-efectivamente computables.*

Esta es la tesis de Church.

Conjetura: **Tesis de Church**
Toda función Σ-efectivamente computable es Σ-recursiva

Si bien no se ha podido dar una prueba estrictamente matematica de la Tesis de Church, es un sentimiento comun de los investigadores del area que la misma es verdadera.

Guía 9: Resultados del Paradigma Recursivo

Esta es la conclusión final de la materia. En esta parte apuntamos a probar unos resultados más sobre los paradigmas que estuvimos estudiando. Varios resultados los aplicaremos específicamente para el paradigma de Gödel pero como los paradigmas son equivalentes, aplican también para los demás.

3.1 Algunas propiedades útiles

Lemma:
$f_1 : D_{f_i} \subseteq \omega^n \times \Sigma^{*m} \to O,\ i = 1, \ldots, k$ son funciones Σ-recursivas tales que $D_{f_i} \cap D_{f_j}$ para cada $i \neq j$.
Entonces la función $f_1 \cup \ldots \cup f_k$ es Σ-recursiva

Lemma: **Restricción de Funciones Σ-recursivas**
Sea $f : D_f \subseteq \omega^n \times \Sigma^{*m} \to O$ es Σ-recursiva y $S \subseteq D_f$ es Σ-recursivamente enumerable, entonces $f|_S$ es Σ-recursiva

Lemma: **Operaciones entre conjuntos Σ-R**
Sea Σ un alfabeto finito. Se tiene que:
- Si $S_1, S_2 \subseteq \omega^n \times \Sigma^{*m}$ son conjuntos Σ-RE entonces $S_1 \cup S_2$ es Σ-RE
- Si $S_1, S_2 \subseteq \omega^n \times \Sigma^{*m}$ son conjuntos Σ-RE entonces $S_1 \cap S_2$ es Σ-RE
- Si $S \subseteq \omega^n \times \Sigma^{*m}$. Si S es Σ-R, entonces S es Σ-RE

RE = Recursivamente enumerable

Lemma:
Sea $S \subseteq \omega^n \times \Sigma^{*m}$. Si S y $(\omega^n \times \Sigma^{*m}) - S$ son Σ-RE, entonces S es Σ-R

Teorema:
Dado $S \subseteq \omega^n \times \Sigma^{*m}$, son equivalentes:
- S es Σ-RE
- $S = I_F$ para alguna $F : D_F \subseteq \omega^k \times \Sigma^{*l} \to \omega^n \times \Sigma^{*m}$ tal que cada $F_{(i)}$ es Σ-R
- $S = D_f$ para alguna función Σ-R f
- $S = \varnothing$ o $S = I_F$ para alguna $F : \omega \to \omega^n \times \Sigma^{*m}$ tal que cada $F_{(i)}$ es Σ-PR

3.2 Halting Problem y los conjuntos A y N

Def: **AutoHalt**

$$\text{AutoHalt}^{\Sigma} = \lambda\mathcal{P}\big[(\exists\, t \in \omega) \quad \text{Halt}^{0,1}(t, \mathcal{P}, \mathcal{P})\big]$$

Lemma:
Si $\Sigma_P \subseteq \Sigma$, entonces AutoHalt^{Σ} **no** es Σ-Recursivo

Teorema:
Si $\Sigma_P \subseteq \Sigma$, entonces AutoHalt^{Σ} **no** es Σ-efectivamente computable

Def: A **y** N

$$A = \{\mathcal{P} \in \text{Pro}^{\Sigma} \ : \ \text{AutoHalt}^{\Sigma}(\mathcal{P}) = 1\}$$

$$N = \text{Pro}^{\Sigma} - A = \{\mathcal{P} \in \text{Pro}^{\Sigma} \ : \ \text{AutoHalt}^{\Sigma}(\mathcal{P}) = 0\}$$

Lemma:
A es Σ-RE y **no** es Σ-R
N **no** es Σ-RE

Proposición:
Supongamos que $\Sigma_P \subseteq \Sigma$. Entonces A es Σ-efectivamente enumerable y no es Σ-efectivamente computable.
El conjunto N no es Σ-efectivamente enumerable. Es decir, A puede ser enumerado por un procedimiento efectivo pero no hay ningun procedimiento efectivo que decida la pertenencia a A y no hay ningun procedimiento efectivo que enumere a N.
Mas aun, si un procedimiento efectivo da como salida siempre elementos de N, entonces hay una cantidad infinita de elementos de N los cuales nunca da como salida

Proposición:
Supongamos que $\Sigma_P \subseteq \Sigma$.
Sea P:

$$P = C_1^{0,1}|_A \ \circ \ \lambda t\alpha[\alpha^{1\dot{-}t}\text{SKIP}^t]|_{\omega\times\text{Pro}^{\Sigma}}$$

Entonces P es Σ-R pero la función $M(P)$ no es Σ-efectivamente computable (y por lo tanto tampoco es Σ-R)
Notar que $M(P) = \neg\text{AutoHalt}^{\Sigma}(\mathcal{P})$